



Base CPU | OCD
Data paths | Error
CENTER FOR EXASCALE SIMULATION OF
COMBUSTION IN TURBULENCE (EXACT)

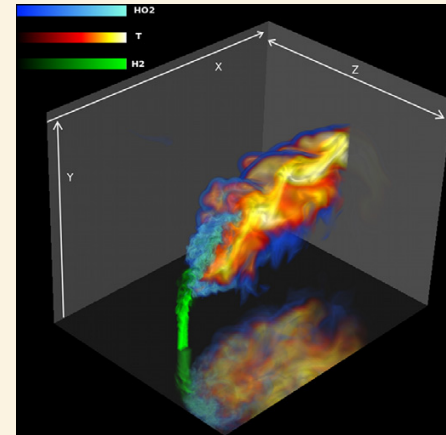
Center for Exascale Simulation of Combustion in Turbulence (ExaCT)

Jacqueline Chen, Director
Combustion Research Facility
Sandia National Laboratories
Livermore, CA

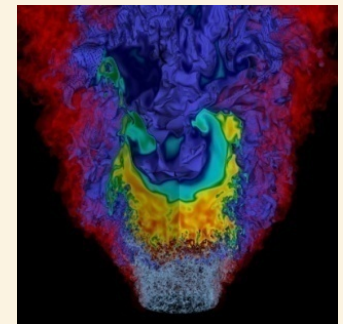
SOS16: Quantifying the Costs of Exascale
Santa Barbara, CA
March 12-15, 2012

Why Combustion

- 83% of U.S. energy comes from combustion of fossil fuels
- National goals
 - Reduce greenhouse gas emissions by 80% by 2050
 - Reduce petroleum usage by 25% by 2020
- Meeting these goal requires a new generation of high-efficiency, low emission combustion systems
 - New designs for IC engines, turbines and burners
 - New fuels
- Examples
 - Engine designs to burn biodiesel for transportation
 - Fuel flexible turbines for power generation
- Why exascale
 - Current design methodologies are largely phenomenological
 - Lack the science base needed to develop new devices / fuels
 - Exascale computing will enable high-fidelity simulations of complex fuels at realistic turbulence and pressure conditions with quantified uncertainty
 - Key element needed to develop and calibrate models needed for engineering design
- Use cases based on increasing pressure and fuel complexity



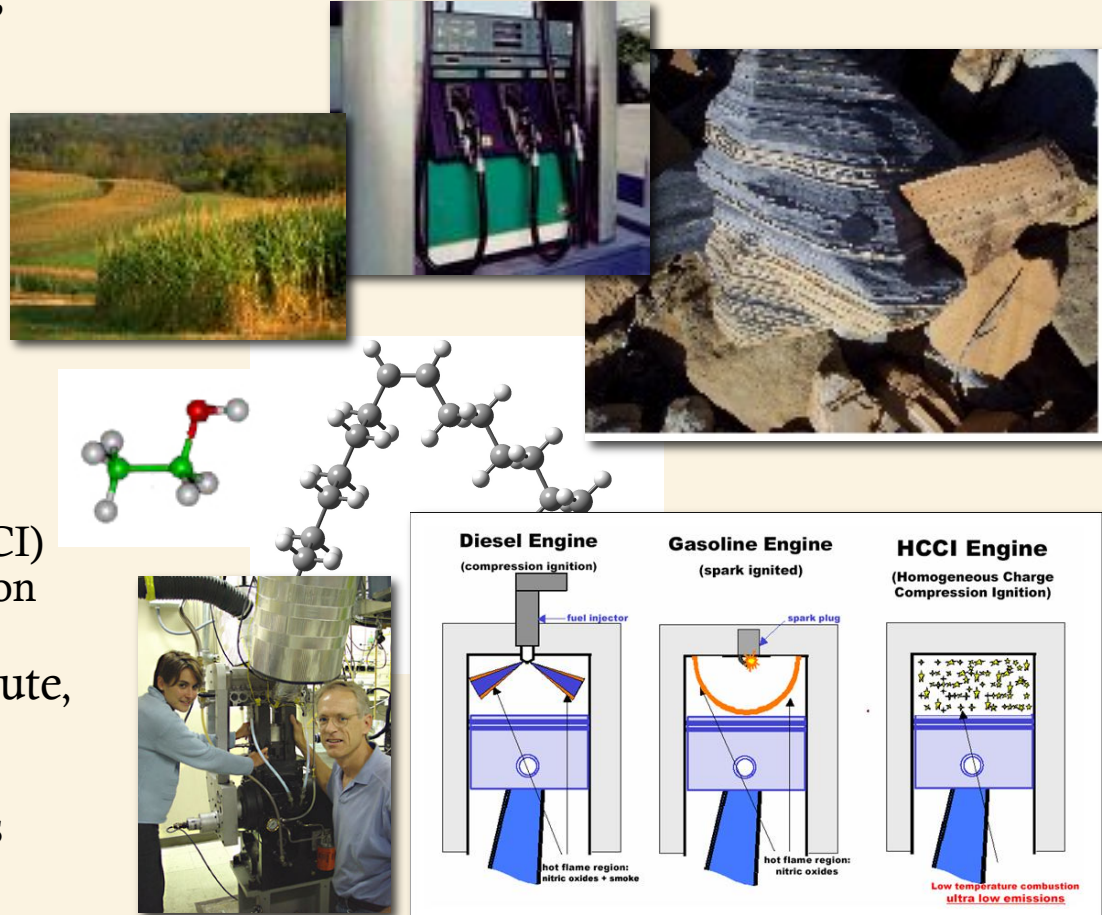
Simulation of a nitrogen-diluted hydrogen jet in crossflow



Simulation of NO_x emissions from a low swirl injector fuels by H₂

Motivation: Changing World of Fuels and Engines

- Fuel streams are rapidly evolving
 - Heavy hydrocarbons
 - Oil sands
 - Oil shale
 - Coal
 - New renewable fuel sources
 - Ethanol
 - Biodiesel
- New engine technologies
 - Direct Injection (DI)
 - Homogeneous Charge Compression Ignition (HCCI)
 - Low-temperature combustion
- Mixed modes of combustion (dilute, high-pressure, low-temp.)
- Sound scientific understanding is necessary to develop predictive, validated multi-scale models!

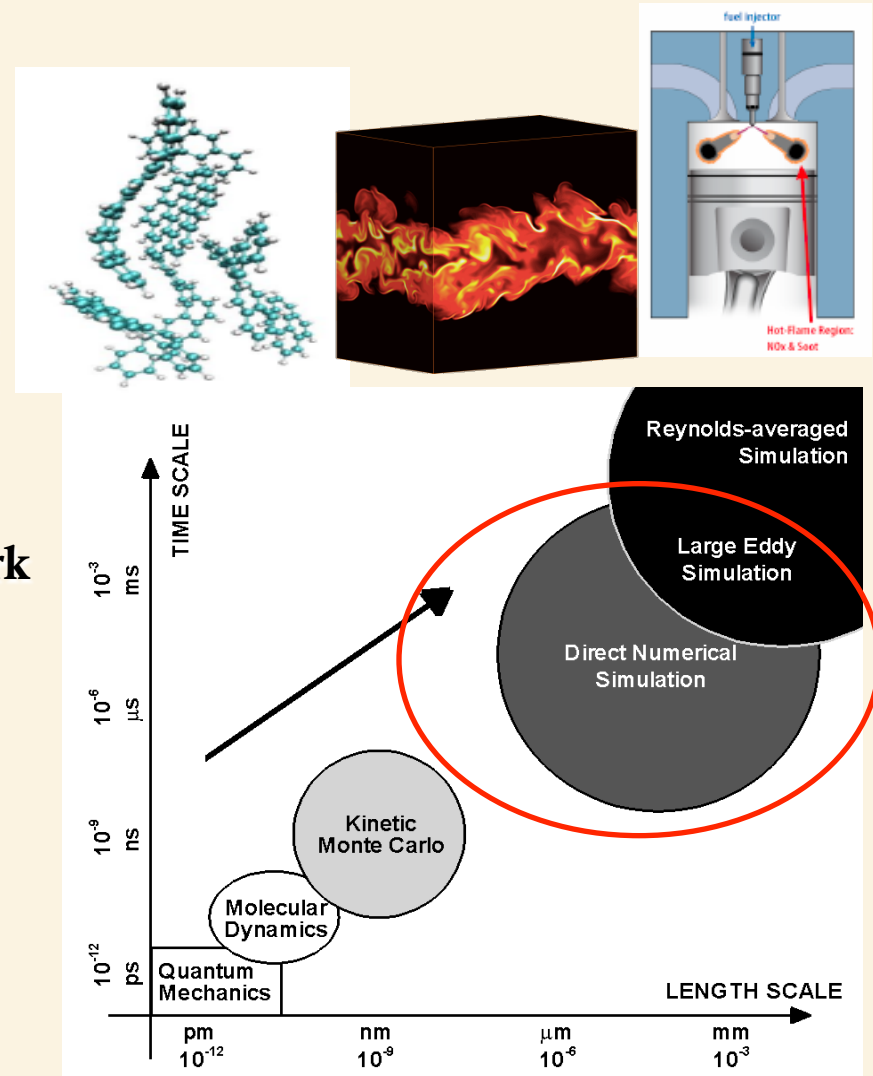


Multi-scale Modeling of IC engine processes

- Multi-scale modeling describes IC engine processes, from quantum scales up to device-level, continuum scales

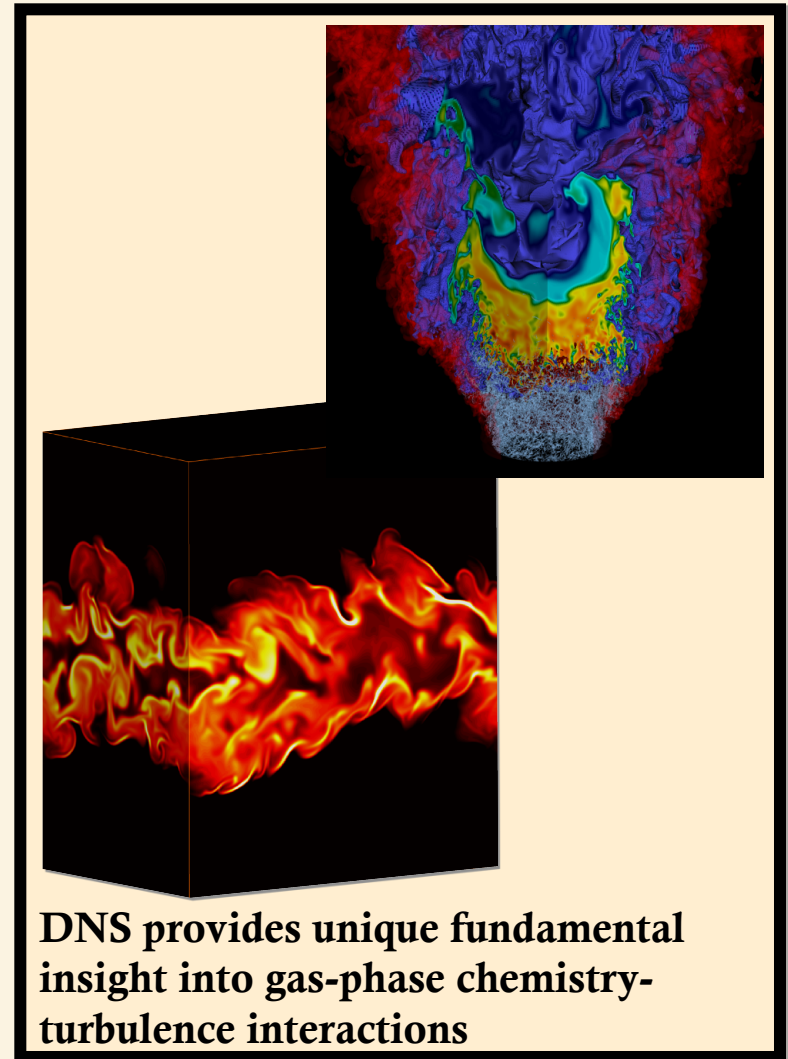
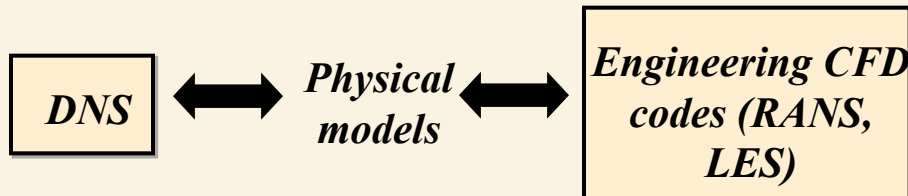
- Needs:

- Develop a general theoretical framework for transfer of information from one scale to the next
- Use HPC to bridge the current gap between coarse-grained atomistic approaches and fine-grained continuum approaches



Direct Numerical Simulation

- Used to perform first-principles-based DNS of reacting flows
- Solves compressible or low-Mach reacting Navier-Stokes equations
- High-fidelity numerical methods (high-order finite difference and AMR)
- Detailed reaction kinetics and molecular transport models
- Multi-physics (sprays, radiation and soot)
- Runs on all major platforms, scales well to 10-20 pflop machines



Co-Design Objectives

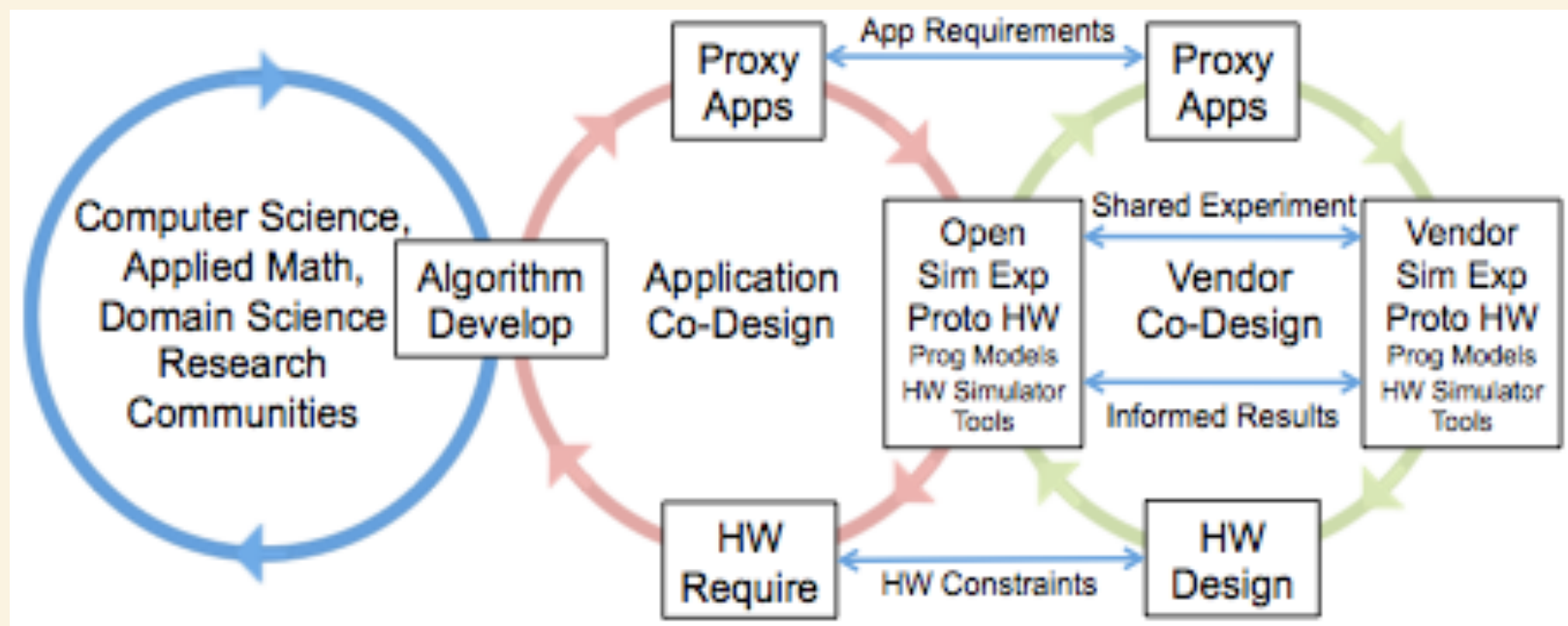
- Goal of ExaCT is to influence the sea-change in architecture and software of next generation systems - driven by power and cost constraints - to be well-suited for combustion simulation
 - Higher concurrency in low-power many-core, possibly heterogeneous nodes
 - Performance based on memory access patterns and data movement, not FLOPS
 - High synchronization costs
 - Reduced memory per core
 - Increased disparity between I/O and compute speed
 - Machine complexity – fault tolerance in all aspects of stack

Major Themes for Combustion Co-design

- **Integrate analysis with simulation**
 - Combustion simulations are data rich
 - Writing data to disk for subsequent analysis is currently near infeasibility
 - Integrate analysis and uncertainty quantification directly into simulation process
 - Makes simulation look much more like physical experiments in terms of methodology
- **Rethink implementation and analysis of basic algorithms in terms of potential architectures**
 - Expose more concurrency
 - Distributed AMR metadata
 - Analysis of algorithms has typically been based on a performance FLOPS paradigm – can we analyze algorithms in terms of a more realistic performance model
- **Develop programming models more suited to new architectures**
 - MPI provides reasonable approach for coarse-grained parallelism but tools at fine-grained level are inadequate
 - We express codes in terms of FLOPS when data layout and data movement control performance
- **Evaluate performance impact of hardware tradeoffs and provide input to vendors**

Co-Design Process

- Iterative co-design loop (joint exploration of hardware and algorithm space)
- Proxy applications: skeletal, node-level (compact apps and compute kernel apps)
- Abstract machine model – open architectural simulators, prototype hardware
- Data abstractions, programming models (DSL's) and supporting runtime
- Bi-directional interface with vendors

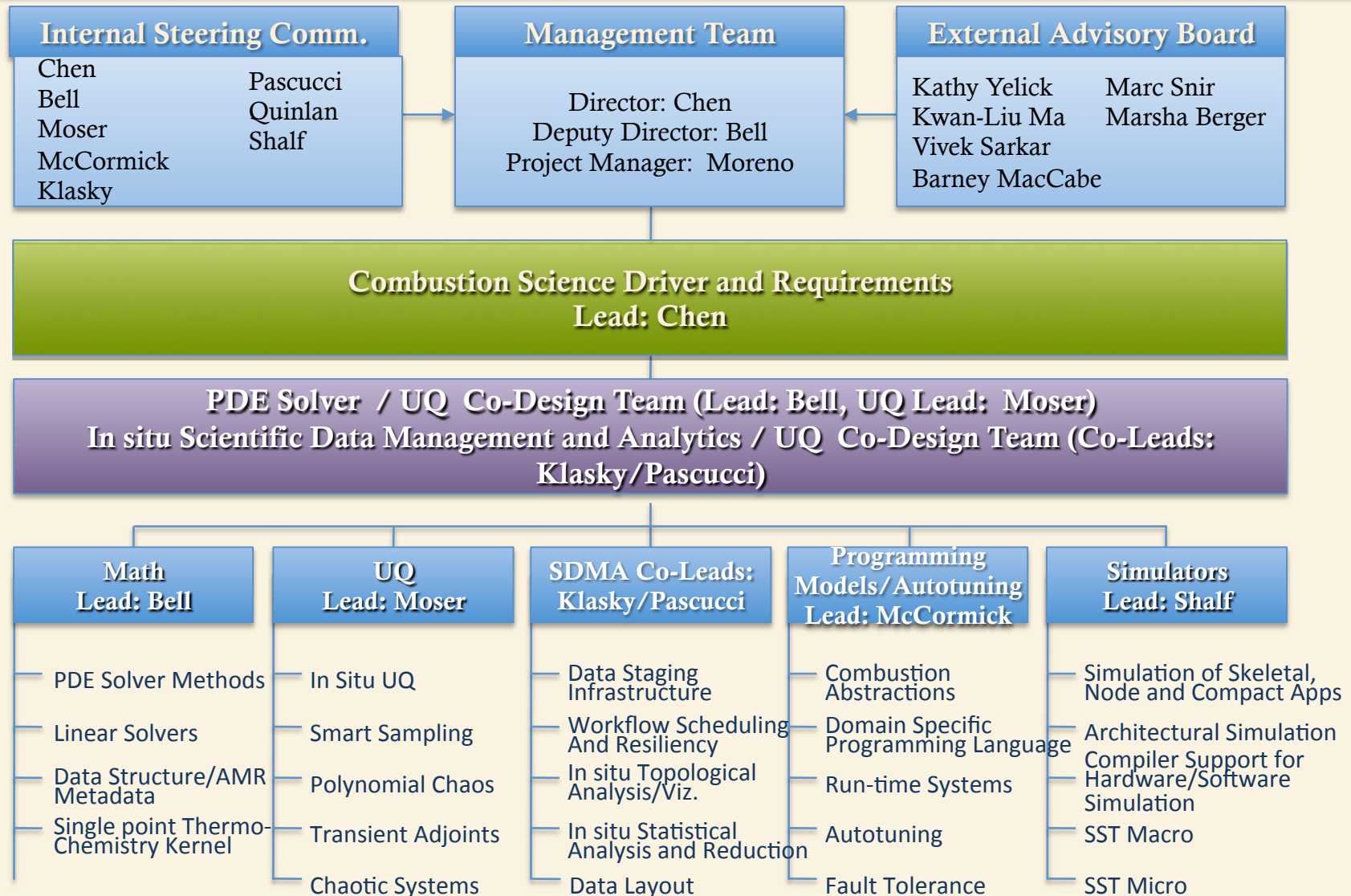


Co-design Consortium Vendor Operation Plan, 2012

CS Aspects

- Programming models
 - Develop programming methodologies that allow us to express the algorithms
 - Expressiveness
 - Performance
 - Execution model
 - Evolve an abstract machine model
 - Capture key aspects of performance
 - Simple enough to inform algorithm development
 - Fault tolerance
- Simulators
 - Tests of alternative architectural features
 - Develop to meet co-design theme requirements as needed
- Bi-directional interface to vendors

ExaCT Organizational Chart

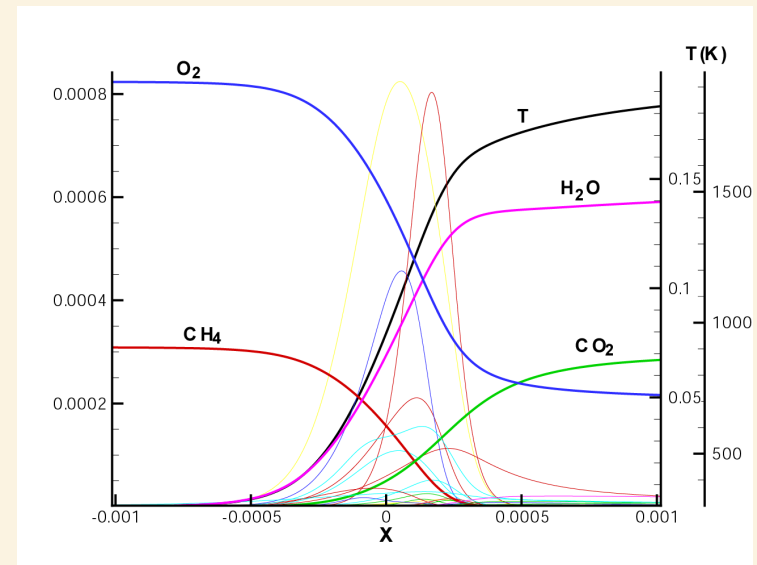


Highlights from Cross-cut Areas

- PDE Solver Algorithms/Implementations
- Intrusive UQ
- SDMA and *in situ* Analytics/Viz.
- Programming Models
- Hardware Simulation

Reacting Flow Simulation Methodology

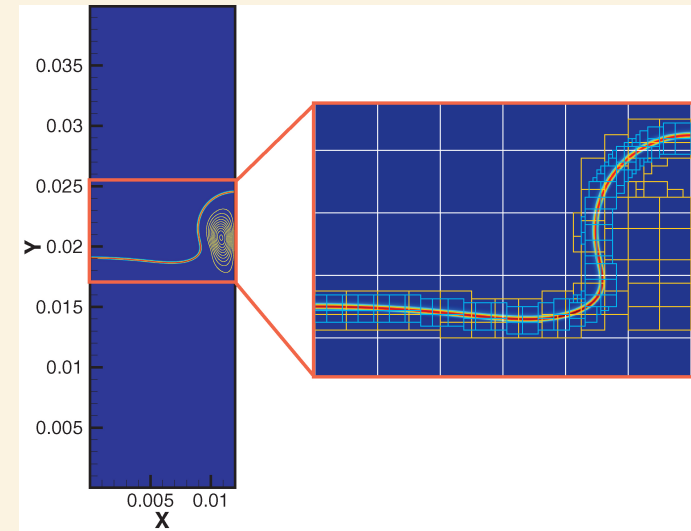
- Physical processes
 - Detailed chemical kinetics
 - Highly nonlinear
 - 10's-100's of species
 - 100's or 1000's of reactions
 - Rich internal flame structure
 - High fidelity species transport
 - Complex nonlinear parabolic systems
 - Turbulent fluid mechanics
 - Compressible, variable viscosity (full stress tensor)
 - Need to resolve several decades of scale
 - Domain \rightarrow integral scale \rightarrow Kolmogorov scale
 - Radiation (optically thin)



Internal structure of methane flame: 53 species; 325 reactions

Simulation Methodology: Combustion Requirements

- Target use cases are high-pressure flames with complex fuels
 - Flames become much thinner as pressure is increased
 - Chemical and transport parameters are not well-known
 - Intermittency (e.g. ignition, turbulence)
- Requirements for methodology
 - Adaptive mesh refinement
 - Turbulence and flames have different resolution requirements
 - Support multiple flow regimes
 - Compressible Navier Stokes
 - Low Mach number formulation
 - Common framework
 - Integrated UQ and *in situ* analytics



Core Exascale Issues

- Basic discretization methodology
 - Reduce memory movement and memory capacity / FLOP
 - Expose more concurrency
 - Express and manage data locality
 - Reduce synchronization
 - Analysis of algorithm in terms of NEW cost-model for abstract machine -- measure algorithmic complexity in terms of costs of memory and data movement instead of FLOPs
- Linear solvers
 - Reduced communication / synchronization
- AMR
 - Hierarchical metadata
 - Data-movement aware operations (include cost of data movement in regridding)
 - Reduced synchronization
- All of this issues need to be address in terms of tradeoffs in architecture
 - Internode network topology
 - Intranode design
 - Individual core characteristics

Interface

- What are the key questions at the interface of math, programming models and hardware?
- Horizontal data locality
 - Control data layout on a node
 - Data layout matched to intranode connectivity / topology
 - Reduce, eliminate or localize cache coherence
 - These require programming model to provide tools to express the relevant constructs
 - Use type-system to express data layout and topology
 - USE DSLs or other higher-level constructs to make locality implicit
- Vertical data locality
 - Control data motion through cache/memory hierarchy (to processor and back)
 - Maximize reuse of data to reduce memory access
 - Requires programming model support
 - Constructs to make software managed memory and explicit data motion easier to use

AMR at the Exascale

- AMR can be viewed as minimizing the number of degrees of freedom needed to represent the solution
 - An *a priori* good match to reduced memory exascale architectures
 - Structured-grid AMR is naturally suited to hierarchical parallelism
 - Domain is covered by large aggregate patches of data
- Need to deal with algorithmic complexity
 - Hierarchical treatment of data metadata
 - Regions of decreasing granularity
 - Only store details about your region and its neighbors
 - AMR retains locality – it's just more complex
 - Regridding and dynamic load balancing more holistic
 - Include all factors in estimating cost of next step
 - Cost of data movement
 - Heterogeneous node performance
 - Restructure integration to reduce synchronization and increase concurrency

Execution Model

- What is a good choice of execution model? How does best choice depend on architectural details?
- Current bulk synchronous approach is probably not the best choice
 - Sources of performance heterogeneity are increasing
 - Examples of heterogeneity include:
 - Thermal-performance throttling
 - Software fault-recovery (creates performance perturbations)
 - Adaptive algorithms and non-uniform cost of chemistry ODE solves
- Asynchronous models to tolerate heterogeneity
 - Asynchronous task execution based model
 - Data flow / functional model
 - Open questions:
 - At what granularity should we express the model?
 - Do algorithms need to be changed to better reflect alternative models

Development of Combustion Proxy Apps

- Refactor compressible integration methodology, S3D, into the BoxLib framework (S3D-Box)
 - Establish baseline common data abstraction
 - Prelude to adaptive implementation
 - Expose underlying data dependencies
 - Facilitates mapping onto alternative node architectures
 - Enable deep dependency analysis for compiler transformations / DSL definition
- Define multigrid compact app
 - Representative of communication intensive parts of algorithm
 - Skeletonization to extract communication patterns
 - Analyze impact of network topology, latency and bandwidth on performance

Proxy Apps, cont'd

- Develop compact apps for physical property evaluation and stencil operations
 - Test different strategies for mapping data onto nodes
 - Basis for evaluating node architecture options
 - Extract key kernels for more detailed analysis
 - Performance tuning/modeling
 - Evaluation of core-level architectural features
- Develop simplified AMR compact app
 - Extract skeleton for analysis of data movement / communication patterns
 - Analyze impact of network characteristics on AMR performance
 - Baseline for refactoring core AMR toolset

Uncertainty Quantification: Goals & Challenges

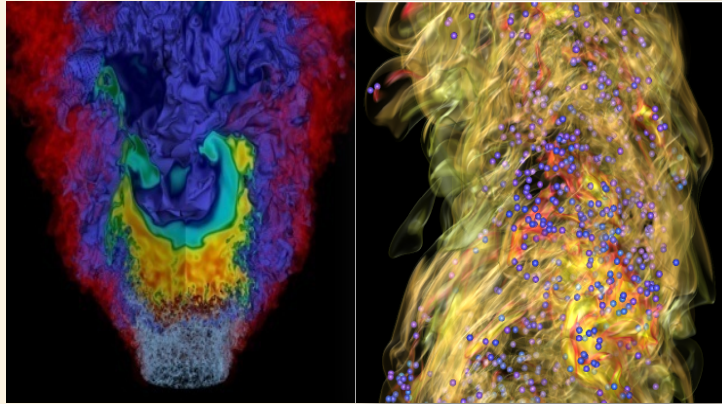
- Science Questions: How are predictions of detailed turbulent flame dynamics (e.g. extinction) and average combustion properties (e.g. NO_x) impacted by uncertainties in chemical reaction models? What new data are needed to reduce these uncertainties?
- UQ computational challenges arise because:
 - Simulations are expensive
 - There are many uncertain parameters
 - Turbulence is chaotic
- Develop the exascale opportunities to address these issues:
 - Use adjoints for sensitivity derivatives, but must evaluate possible time horizon, since linearized solutions grow exponentially
 - Use polynomial chaos expansions but must evaluate utility for chaotic systems

UQ: Methodology & Algorithms

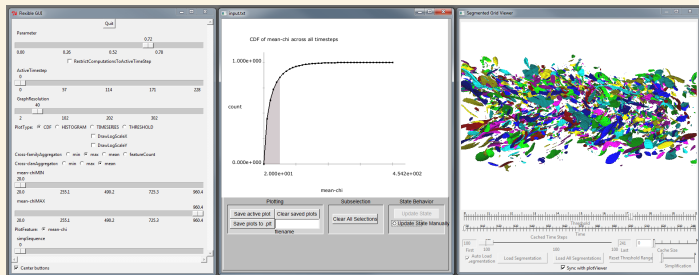
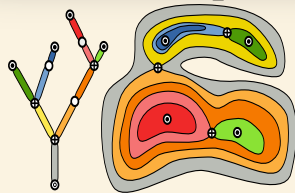
- Adjoint formulation to be integrated in S3D-Box
 - Adjoint-consistent numerical methods
 - Adjoints are computed backward in time, linearized about forward solution
 - Storage and staging of check-point data needed for adjoints (SDMA)
 - Possibility to perform adjoint analysis in local space-time domains to save storage (SDMA)
- Polynomial chaos formulation implementation in S3D-Box
 - Low-level software support for PCE calculations (DSL's)
 - Can attain high flop/memory access
 - Address numerical issues with linear and non-linear solvers
 - Assessment of convergence of PCE representation

SDMA Technologies

Volume rendering can be performed *in situ* using < 1% of simulation time, providing insight into variable interactions.

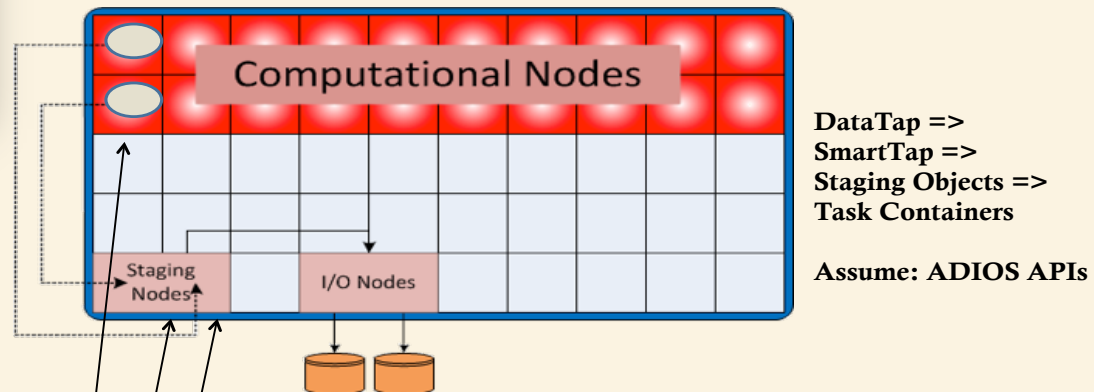


Topological methods provide compact representations, creating drastic data reductions



Hybrid staging allows visualization and analysis components to be easily programmed so that some are executed on cores where the application is running, and some are executed on a remote staging area, which is all on the same HPC resource.

Hybrid Staging – Basics



In Situ Processing Resources:

- (1) InLine or Helper Cores
- (2) Staging Nodes (+GPUs, NVRAM)
- (3) Offline – post storage
- (4) Remote and Cloud Resources

Execution Model:

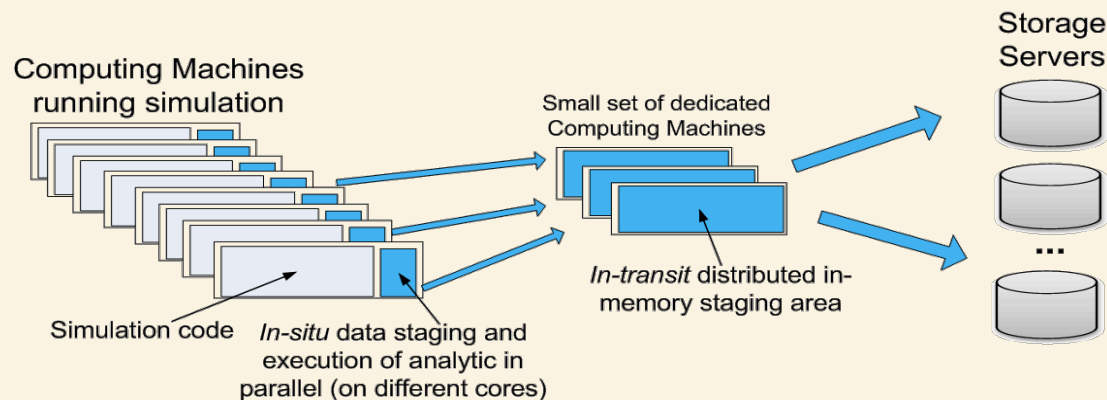
- (1) I/O Graphs: task-based analysis/viz
- (2) Asynch. Data Movement: IB, Portals, ...
- (3) Managed Execution: Containers, ...
- (4) Cloud Access/Usage: S3/EC2 bindings

SDMA High Level Goals

- Co-design the knowledge discovery process for exascale combustion science
- Reduce data movement required for downstream analysis & exploration
 - Characterize features of interest efficiently using topological analysis techniques
 - Visualize data with embedded topological features
- Abstract I/O, visualization and analysis through a componentized middleware and utilize a *SOA* approach to accelerate *in transit* tasks
- Study communication and node-level behaviors and understand tradeoffs for the SDMA suite of routines
- Integrate DSL/programming model abstractions into visualization & analytics codes to abstract hardware-specific optimizations
- Support *in transit* processing for integration of UQ within the SDMA infrastructure

S3D with Staging

- ADIOS is being implemented as a Service Oriented Architecture
 - Create an environment where applications can abstract data movement/IO as read/writes/queries



- Exploit complex-memory hierarchies with *Hybrid Data Staging* to:
 - Decrease the gap between CPU and IO speeds
 - Dynamically deploy and execute data analytical or pre-processing operations either in-situ or in-transit
 - Improved IO write performance

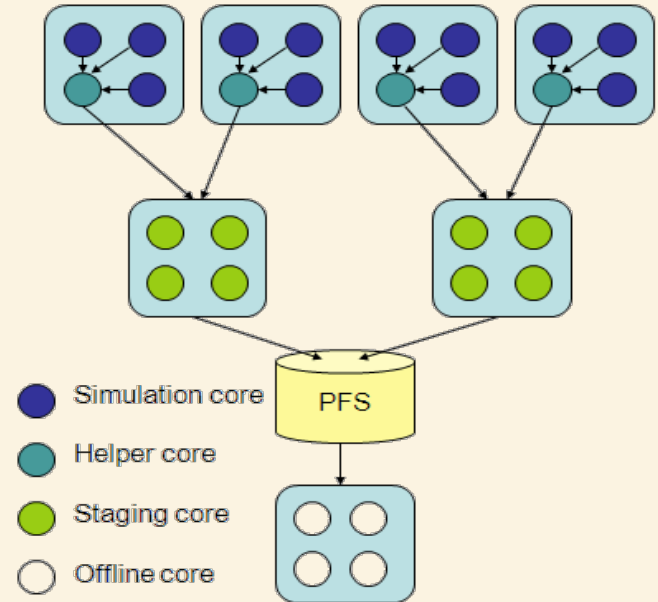
Co-design the Analytics Pipeline

- **Optimization:**
- **Where/How should analytics be run?**

- Inline with simulation?
- Separate cores? Helper cores?
- Separate staging nodes?
- Offline?

- **Guided by what metrics?**

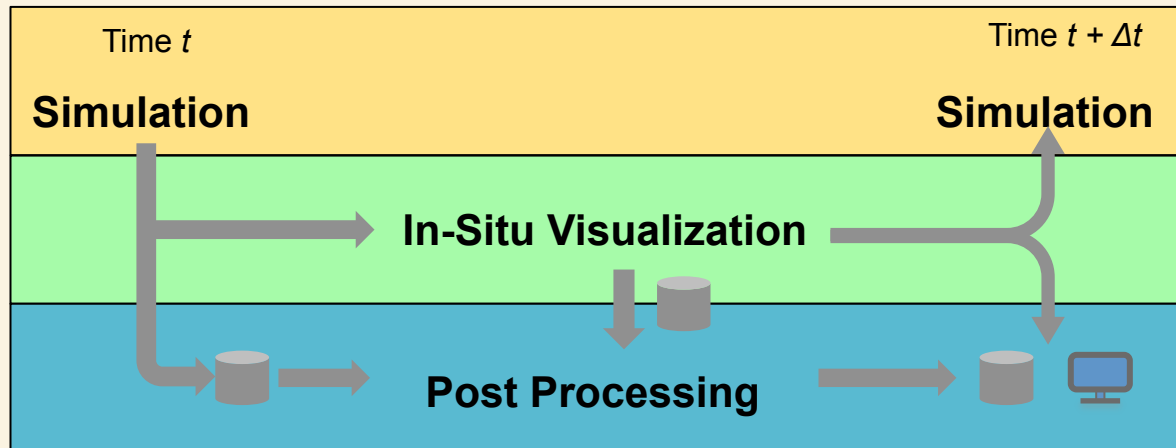
- *Performance*: **Total Execution Time** of both simulation and analytics
- *Cost*: **CPU hours** charged for simulation and analytics
- *Time to Data*: **Delay** between data generation and analytics results
- *Data movement* ~ **Power** consumption



Visualization Within SDMA

- Goals:
 - Define initial compact applications
 - Representative parallel visualization algorithms for simulations, UQ, and topological analysis
 - Connect visualization with SDMA system as auxiliary service
- Research Topics:
 - Mechanisms for visualization routines to access data from simulations, UQ, and topological analysis
 - Mechanisms for visualization routines to save and transfer results
 - Characterization of data movement and communication of visualization routines within SDMA
 - Programming model
 - Architecture simulation

Previous In-Situ Visualization Implementation



Simulation



```
void s3drender_init_(
  int *myid, int *gcomm,

  double *species,
  char *speciesNames,

  double *loc,

  double *x, *y, *z,
  int *nx, *ny, *nz,
  int *npx, *npy, *npz,

  int neighbors[6])
```

MPI Communicator

pointer to local scalar variable

pointer to local particle data



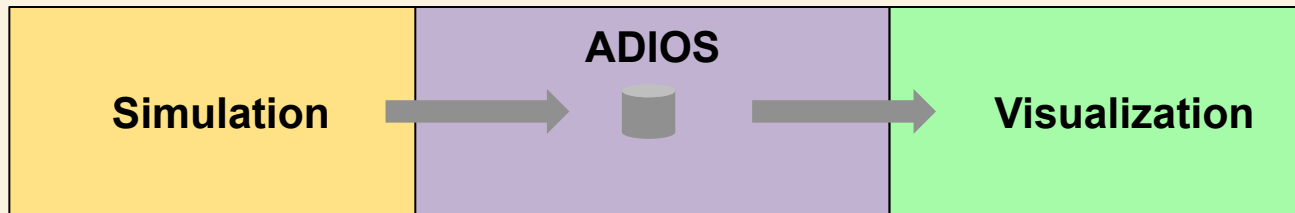
**size and coordinates of
global domain
and local partition**

neighbor processors



Visualization

Current Initial Integration Effort



```
s3d_adios_render_init() {  
  ADIOS_FILE *f = adios_fopen(...);  
  ADIOS_GROUP *g = adios_gopen(...);  
  
  ADIOS_VARINFO *v = adios_inq_var(...);  
  
  v->dims[0], v->dims[1], v->dims[2];  
  
  adios_read_var(...);  
  
  ...  
}
```

ADIOS file and group handles

ADIOS variable information

size of global domain

pointer to local variable

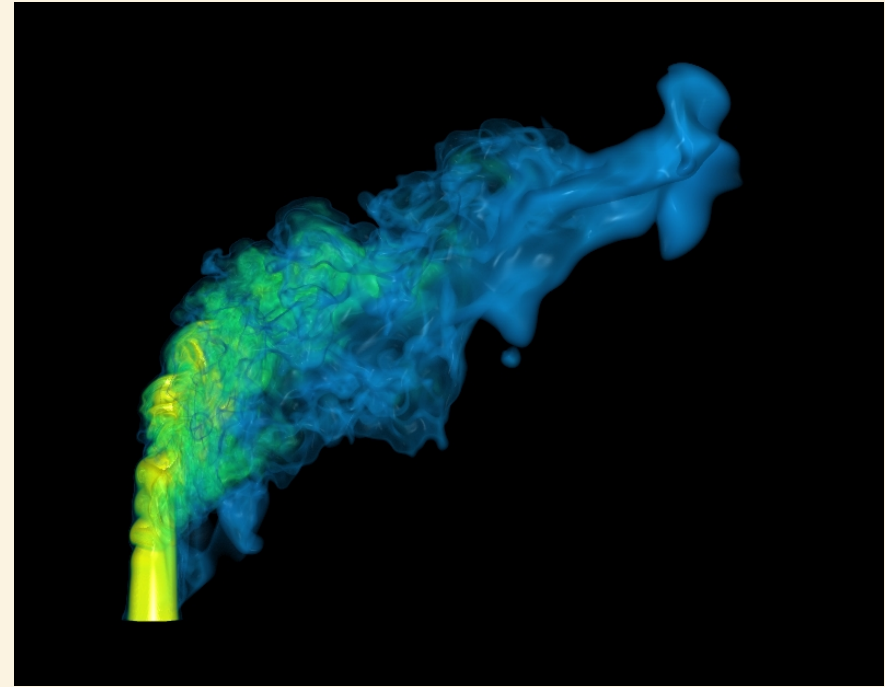
**MPI communicator and
domain partition are decided
by visualization**



Visualization

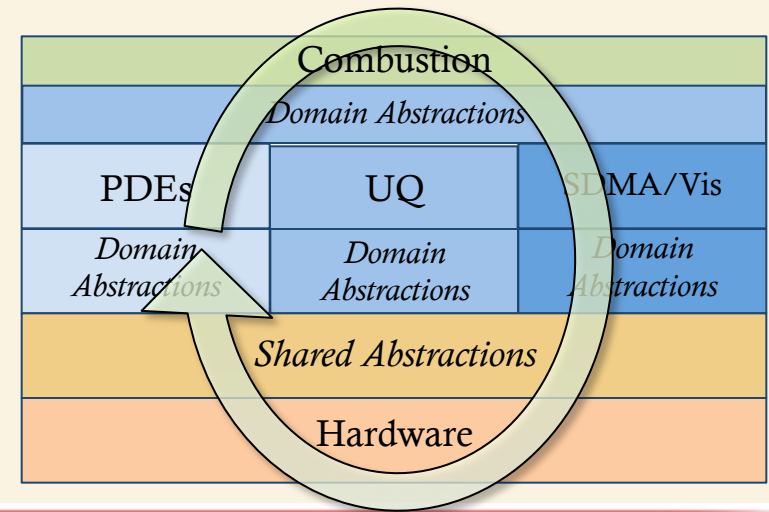
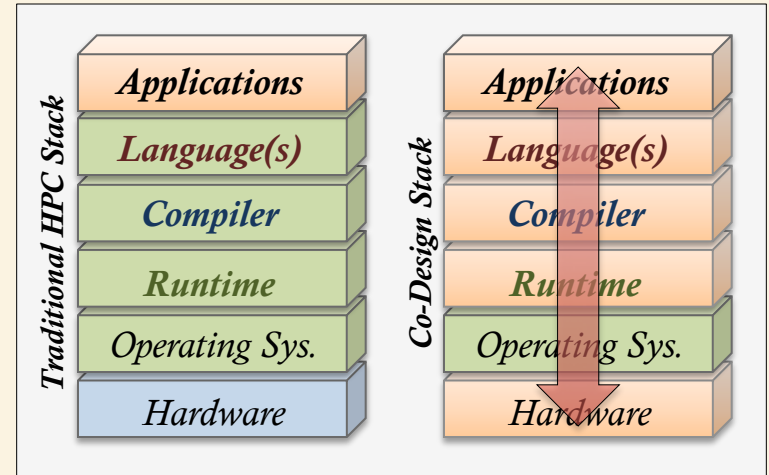
Current Initial Integration Effort

- Visualization Result
 - Simulation: Jet-in-Cross-Flow
 - Variable: H2
 - Domain Grid: 1408x1080x1100
 - Directly render data from ADIOS BP file through ADIOS APIs



Programming Model Overview

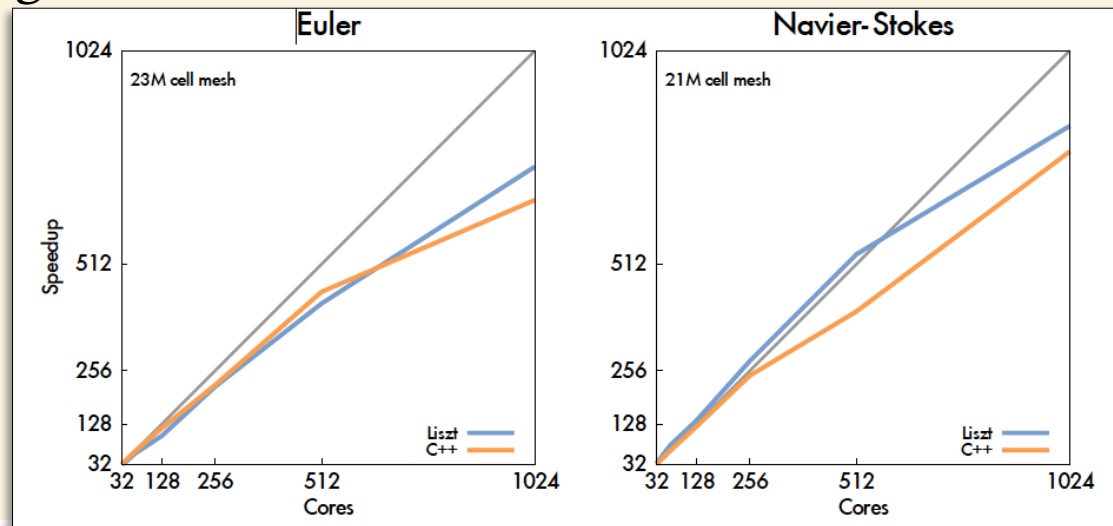
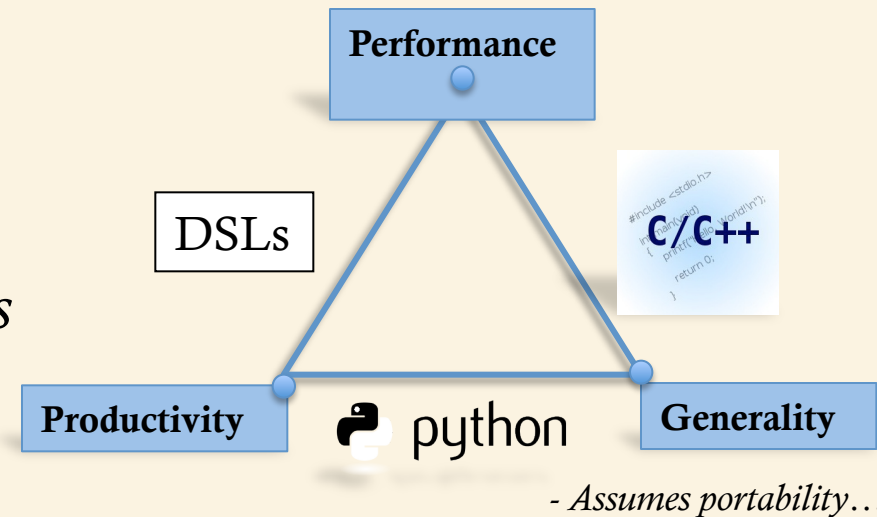
- “Abstraction-driven” co-design
 - Fundamental in the design of *domain-specific languages* (DSLs) and supporting runtimes
 - Iterative process from the hardware “up” and the application “down”
- Incorporation of auto-tuning to produce better code
- Development of compact-apps key to co-design of abstractions and understanding implications of architectures



Programming Models

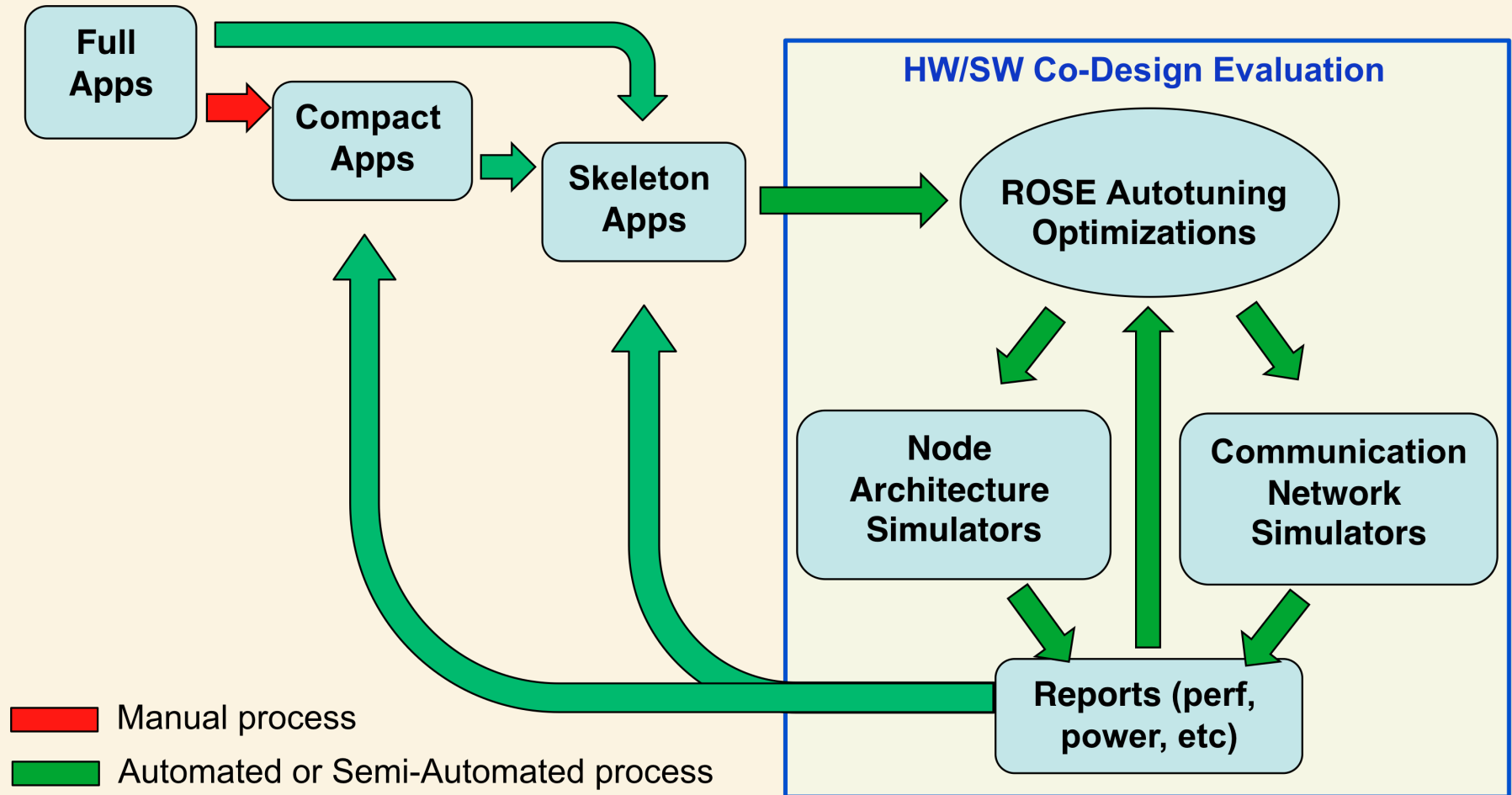
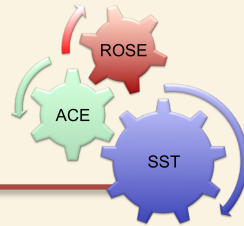
Overview – Domain-Specific Languages

- Fundamental trade-off is generality vs. specialization
- Focus on embedded DSLs: *a language that in some fashion extends an existing general-purpose language*
- Example: Liszt (Stanford)
 - Single source, multiple targets
 - See SC11 paper



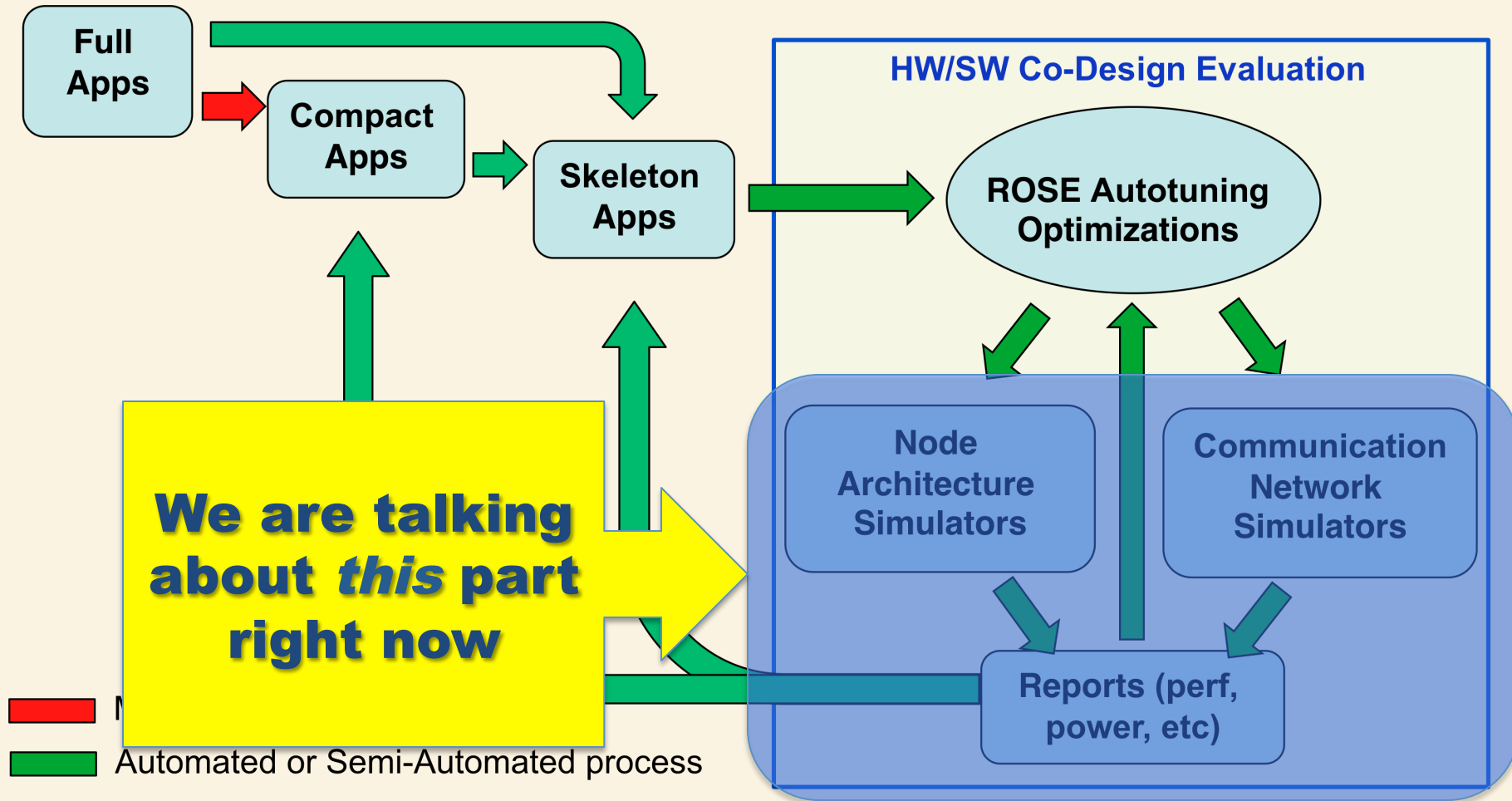
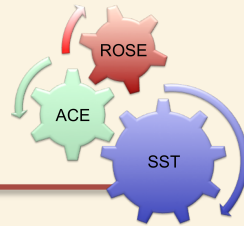
Overall CoDesign Flow

Automatic Generation of Skeletons for Rapid Analysis



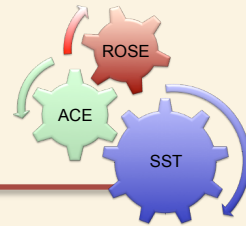
Overall CoDesign Flow

Automatic Generation of Skeletons for Rapid Analysis



CECDC CoDesign Tools Overview

Architectural Simulation to Accelerate CoDesign



ROSE Compiler: Enables deep analysis of application requirements, semi-automatic generation of skeleton applications, and code generation for ACE and SST.

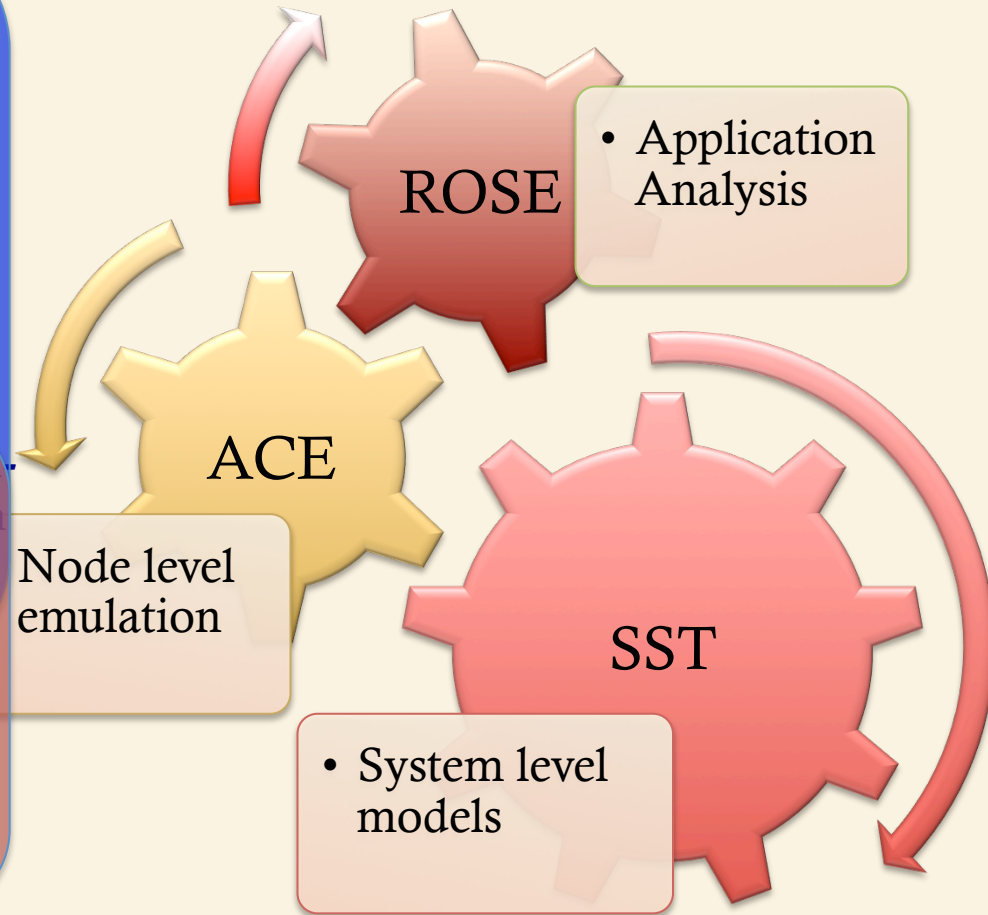
ACE Node Emulation: Rapid design synthesis and FPGA-accelerated emulation for rapid prototyping cycle accurate models of manycore node designs.

ASCR-funded Simulation Infrastructure Project

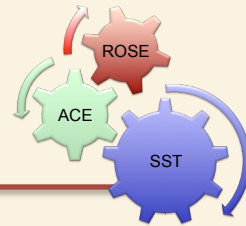
SST Macro System Simulation: Enables system-scale simulation through capture of application communication traces and simulation of large-scale interconnects.

SST: Structure Simulation Toolkit

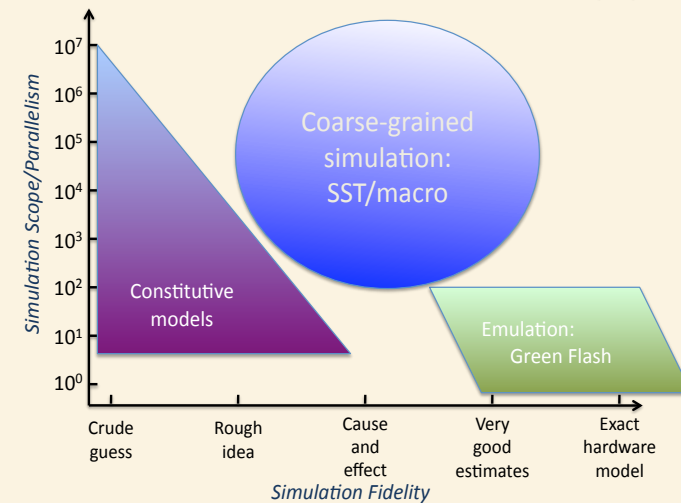
NNSA-funded Simulation Tools
SST Micro Software Simulators: Software simulation for node-level simulation
(ASC Program)



Role of Architectural Simulation



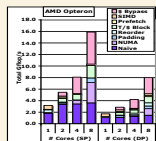
- Simulate hardware *before* it is built!
- Break slow feedback loop for system designs
- Protect vendor IP
- *Insert applications and algorithms into the tightly coupled hardware CoDesign process*



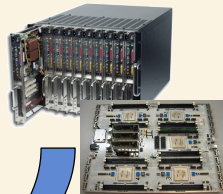
**Design New System
(2 year concept phase)**

Synthesize SoC (hours)

**Tune
Software
(2 years)**

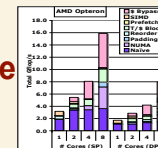


**Cycle Time
4-6+ years**

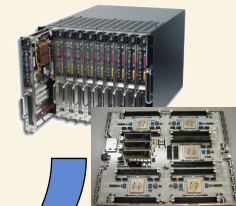


**Build
Hardware
(2 years)**

**Autotune
Software
(Hours)**



**Cycle Time
1-2 days**



**Emulate
Hardware
(RAMP)
(hours)**

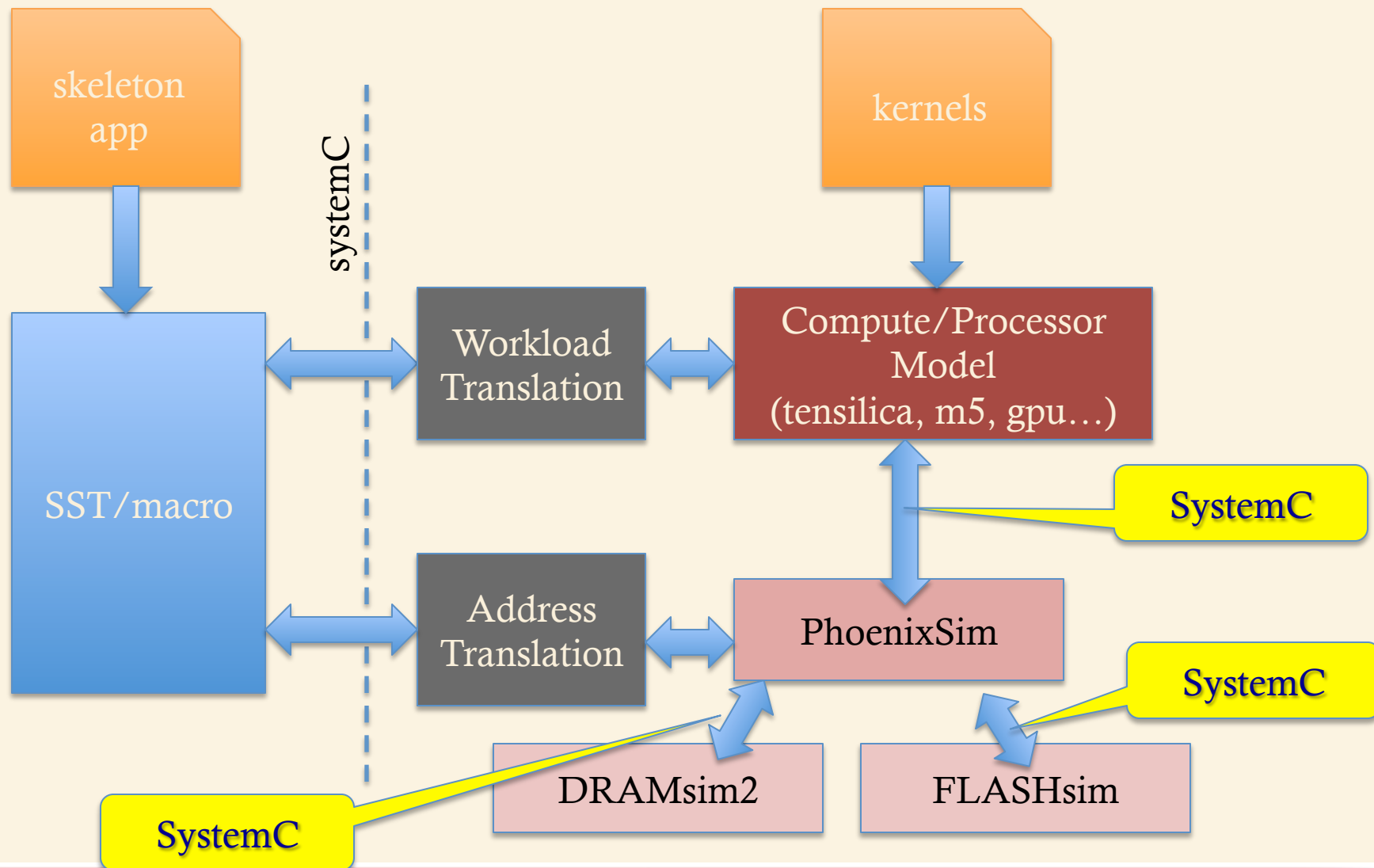
Port Application

Build application

Architectural Simulation Plan

- **Validate Simulation Infrastructure**
 - We are using predictive simulation capability to understand the consequence of hardware/software trade-offs in co-design
 - Therefore must ensure it is correctly predicting hardware performance and energy (V&V)
- **Interconnect Simulation**
 - Collect communication traces to understand existing codes
 - Generate communication skeletons to enable extrapolation of traces for future systems
- **Node Simulation**
 - Bring up proxy apps (S3D & LMC kernels and compact apps) on simulator compilers
 - Study opportunities for integrating more computation in memory system and scalable alternatives for cache coherence and consequences to language design
- **Crosscutting issues**
 - Establish Vendor Interactions and collaborations
 - Develop common set of metrics and parameters for Abstract Machine Model for target exascale systems (*define in Abstract Machine Model “Living Document”*)
 - Integration of Node simulation with Interconnect simulation
 - Integration/leverage other DOE-funded Exascale Research (Execution Models, Data Movement Dominates, CoDEx, SUPER)

Integrated Simulation Tools Environment

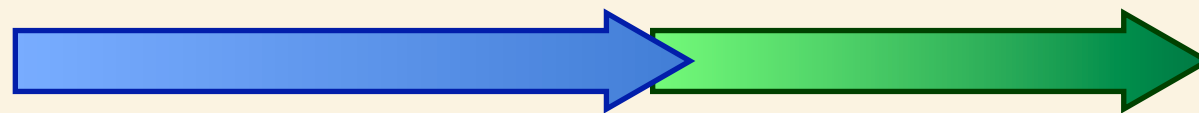
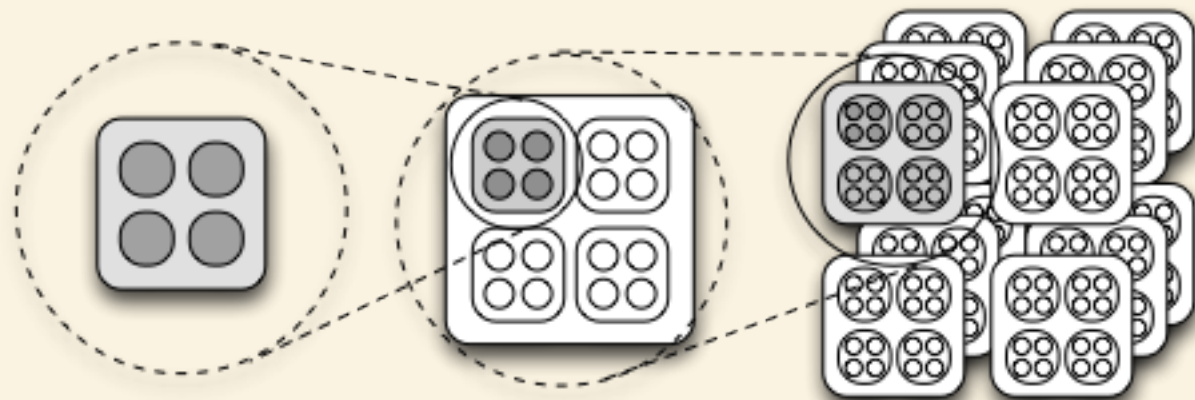


Flavors of Structural Simulation Toolkit (SST)

Single Processor

Whole Node

Entire Machine



SST Micro Simulator
(Processor Core, Memory,
single-GPU)

SST Macro Simulator
(Multi-node, Interconnect)

Initial SST Results

- Focus on initial serial compact apps in SST Micro
- Serial S3D compact app running on GEM5 simulator
 - AMD 2.90GHz K-10 core, A8-3850 (2011) Processor
 - Code compiled with gfortran 4.4, -O3
 - Simulated runtime - 2.609 seconds
 - Benchmarked runtime - 2.857 seconds
 - Error - 8.68%

Initial SST Results

- SST Micro simulator results show:
 - Very low L1, L2 miss rates - less than 0.1%
 - Instruction breakdown to CPU ints:
 - Integer arithmetic - 55.32%
 - Floating-point - 18.15%
 - Memory reads - ~20%
 - Memory writes - ~6.46%

Approaches to Managing Vendor Interactions

- **The “Wall”**
 - Throw compact apps (*and other items*) over the wall for vendors to consider
 - Or vendor asks us to answer questions like “how big should the cache be?” or evaluate feature to help with “sort” operations (*normally a cache-buster*)
- **IP Firewall**
 - One person per vendor interaction gets deep-NDA (*then permanently contaminated*)
- **Collaborate with the “*research division*”**
 - Resolving who owns patent rights
 - Still danger of IP contamination
- **Public/Private repeatable experiments (we are building towards this model)**
 - Use open simulators + compact apps to do experiments in the open
 - Provide those apps + simulation conditions to vendors so they can repeat the experiment using their internal resources
 - If results agree, we can influence their design choices
 - If they disagree, they can identify what we are missing without exposing IP
- **Component Sharing: (can they share their “secret sauce” as black-box?)**

Thanks!

- Questions? jhchen@sandia.gov
- Stay tuned for our website in mid-April:
exactcodesign.org

